monotonicity of multiplication

Johan G. F. Belinfante
2002 September 1

```
<< goedel52.p27; << tools.m

:Package Title: goedel52.p27        2002 September 1 at 2:05 p.m.

It is now:  2002 Sep 1 at 21:37

Loading Simplification Rules

TOOLS.M                    Revised 2002 August 30

weightlimit = 40
```

■ Summary

A monotonicity property of multiplication is derived in this notebook. The usual statement of monotonicity involves three variables, but it is only necessary to require that one of these be a natural number. A relational version of the monotonicity theorem involving only one explicit variable is also derived, as well as useful formulas for a special case.

■ Deriving the relational formulation of monotonicity

The first step just helps to clean things up a bit:

```
SubstTest[subclass, image[z, x], range[z], z -> NATADD]

subclass[image[NATADD, x], omega] == True

subclass[image[NATADD, x_], omega] := True
```

This temporary rule is helpful:

```
SubstTest[subclass, composite[u, id[v], w], composite[u, w],
    {u -> NATADD, w -> composite[inverse[FIRST], NATMUL, LEFT[x]]}]

subclass[composite[NATADD, id[v], inverse[FIRST], NATMUL, LEFT[x]],
  composite[S, NATMUL, LEFT[x]]] == True

subclass[composite[NATADD, id[v_], inverse[FIRST], NATMUL, LEFT[x_]],
    composite[S, NATMUL, LEFT[x_]]] := True
```

The **Assoc** test is used to derive a relational formulation of monotonicity:

```
Map[subclass[#, composite[S, NATMUL, LEFT[x]]] &,
 Assoc[composite[NATMUL, LEFT[x]], NATADD, inverse[FIRST]]]

subclass[composite[NATMUL, LEFT[x], S, id[omega]], composite[S, NATMUL, LEFT[x]]] == True
```

```
subclass[composite[NATMUL, LEFT[x_], S, id[omega]],
  composite[S, NATMUL, LEFT[x_]]] := True
```

■ A more familiar formulation of monotonicity

To obtain a more familiar formulation of monotonicity, we introduce two extra variables. The first of these extra variables is introduced in the following temporary rewrite rule:

```
SubstTest[subclass, composite[u, id[v], w], composite[u, w],
    {u -> composite[NATMUL, LEFT[x]], v -> singleton[y], w -> composite[S, id[omega]]}]

subclass[cart[intersection[omega, P[y]], singleton[natmul[x, y]]],
  composite[NATMUL, LEFT[x], S]] == True

subclass[cart[intersection[omega, P[y_]], singleton[natmul[x_, y_]]],
  composite[NATMUL, LEFT[x_], S]] := True
```

This first temporary rule is used to derive a second temporary rewrite rule of a rather similar nature:

```
SubstTest[implies, and[subclass[u, v], subclass[v, w]], subclass[u, w],
    {u -> composite[NATMUL, LEFT[x], id[singleton[y]], S, id[omega]],
        v -> composite[NATMUL, LEFT[x], S, id[omega]],
        w -> composite[S, NATMUL, LEFT[x]]}]

subclass[cart[intersection[omega, P[y]], singleton[natmul[x, y]]],
  composite[S, NATMUL, LEFT[x]]] == True

subclass[cart[intersection[omega, P[y_]], singleton[natmul[x_, y_]]],
  composite[S, NATMUL, LEFT[x_]]] := True
```

Another new variable is introduced in the final step:

```
SubstTest[implies, subclass[u, v],
 subclass[image[u, singleton[x]], image[v, singleton[x]]],
    {u -> cart[intersection[omega, P[y]], singleton[natmul[z, y]]],
        v -> composite[S, NATMUL, LEFT[z]]}]

or[not[member[x, omega]], not[subclass[x, y]],
  subclass[natmul[x, z], natmul[y, z]]] == True
```

This is the traditional formulation of monotonicity for multiplication of natural numbers:

```
or[not[member[x_, omega]], not[subclass[x_, y_]],
  subclass[natmul[x_, z_], natmul[y_, z_]]] := True
```

■ Comments

The hypothesis **member[x,omega]** is needed because a subclass of a natural number need not be a natural number. If this hypothesis were omitted, the following would provide a counterexample:

```
or[not[subclass[x, y]], subclass[natmul[x, z], natmul[y, z]]] /.
 {x -> singleton[singleton[0]], y -> succ[singleton[0]], z -> 0}

False
```

On the other hand, if either **y** or **z** fails to be a natural number, the conclusion becomes the harmless statement **subclass[V,V]**.

■ Corollary: a special case

An important special case obtained by setting one of the variables equal to **1 = singleton[0]** :

```
SubstTest[implies, and[member[z, omega], subclass[z, x]],
    subclass[natmul[y, z], natmul[y, x]], z -> singleton[0]]
```

```
or[not[member[0, x]], subclass[y, natmul[x, y]]] == True
```

This says that if one multiplies a given natural number by a nonzero number, the result is greater than or equal to the given one.

```
or[not[member[0, x_]], subclass[y_, natmul[x_, y_]]] := True
```

Rewriting this special result as a relational formula involving only one variable requires a certain amount of cleverness. One way to do this is by building in the condition **member[0,x]** as follows:

```
class[w, member[0, x]]
```

```
image[V, intersection[x, singleton[0]]]
```

The factor **id[image[V,intersection[x,singleton[0]]]]** is equal to **Id** or **0** depending on whether or not **member[0,x]** holds; it is used to construct a class that is empty:

```
intersection[composite[id[image[V, intersection[x, singleton[0]]]],
    NATMUL, LEFT[x]], complement[S]] // VSNormality
```

```
composite[id[image[V, intersection[x, singleton[0]]]],
  intersection[complement[S], composite[NATMUL, LEFT[x]]]] == 0
```

The final step converts this into a statement:

```
Map[equal[0, #] &, %]
```

```
or[not[member[0, x]], subclass[composite[NATMUL, LEFT[x]], S]] == True
```

```
or[not[member[0, x_]], subclass[composite[NATMUL, LEFT[x_]], S]] := True
```